Creating Hardware Handler in C/C++ for Agilent TestExec SL

Application Note

Overview on Hardware Handlers

A hardware handler lies in between the TestExec SL and the driver for a hardware module. The purpose of having a hardware handler is to enhance TestExec SL's ability to control devices by communicating directly with the instrument's driver. In other words, the handler contains codes that implement the functions called by TestExec SL when it interacts with the hardware. Hardware handlers can be written in C/C++ or any of the managed (based on Microsoft's .NET software technology) programming languages, including C# and VB.NET.

There are distinctive advantages of using the hardware handler. Firstly, hardware handler offers ease of maintenance in terms of code development. Since function calls are generally the same across instruments, it is easier to reuse actions for different types of test systems. Subsequently, tests or testplans built from the same actions can also be reused. As a minimum prerequisite, all hardware handlers require general purpose functions that are compatible with various kinds of hardware modules:

- Open (initialize) a module.
- Close a module.
- Reset a module to a default state (which can be different from its initialized state).
- Declare any parameters needed to create a unique instance of the handler, such as which instrument identifier to use.

The second advantage lies in value-added features that a hardware handler provides. Instead of only utilizing the function calls for a particular instrument, hardware handlers allow you to perform these activities:

- Insertion of error handling – in the events of failure such as instrument fails to initiate, an exception can be raised to alert the user.
- Support for multiple instruments – hardware handlers can be written as a wrapper to support multiple instruments, such as different models of digital multi-meters. This method eliminates the need to create different handlers for each and every instrument.
- Check for compliance – Additional checking can be appended into the hardware handler to check whether the input data adheres to the data types required.

The steps in creating a hardware handler are:

1) Setting up environment for a hardware handler
2) Creating implementation file
3) Writing the hardware handler
4) Building the project

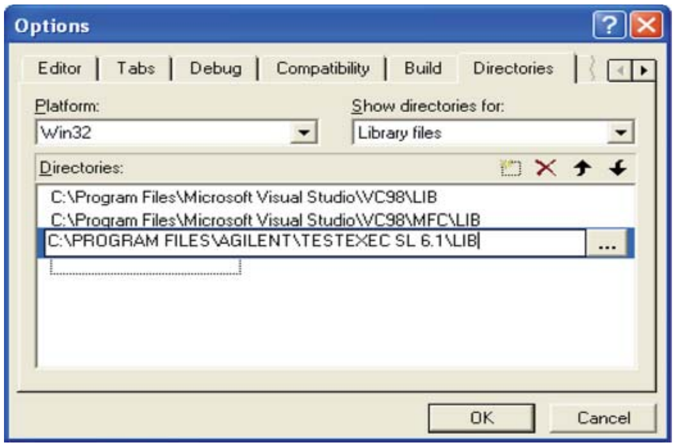
Agilent Technologies

Step 1: Setting up Environment for a Hardware Handler

A prerequisite to creating hardware handler is to first setup the path in your development environment. This example uses the Microsoft Visual Studio C++ 6.0 as the development environment, thus if you are using another C/C++ development environment, the details will vary but the concepts remain similar.

There are two paths that you must set, and these are the paths to access library files and include files. Go to Tools > Options in the Visual C++ menu bar. In the 'Options' box, choose the 'Directories' tab and specify the paths.

Depending on where the installation of TestExec resides in your system, your paths may vary from the example shown here.



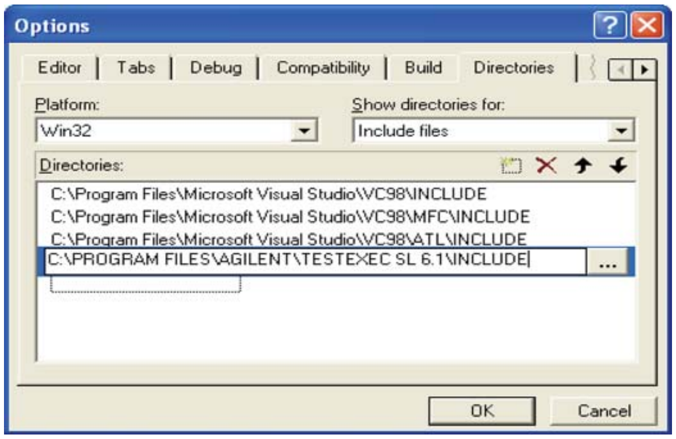
Platform: Win32

Show directories for: Library files

Directories:

```
C:\Program Files\Microsoft Visual Studio\VC98\LIB
C:\Program Files\Microsoft Visual Studio\VC98\MFC\LIB
C:\PROGRAM FILES\AGILENT\TESTEXEC SL 6.1\LIB
```

Figure 1. Setting path for library files.



Platform: Win32

Show directories for: Include files

Directories:

```
C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
C:\PROGRAM FILES\AGILENT\TESTEXEC SL 6.1\INCLUDE
```

Figure 2. Setting path for include files.

Note: *If you are using the TS-5400 library, you have to set the paths for both the library and include files as well.*

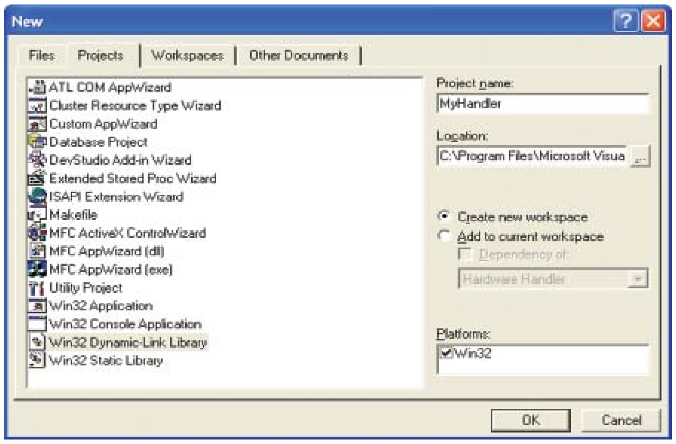
Note: *If you are using the IVI-COM driver, you have to set the paths for both the library and include files as well.*

Step 2: Creating Implementation File

To begin creating a hardware handler, you must first create a new dynamic link library (DLL) project in the Visual C++ by selecting 'Win 32 Dynamic Link Library'. A DLL is a module that contains functions and data that can be used by another module which maybe an application or another DLL. The distinctive benefit of using DLLs is the availability of shared libraries such as modularity. Modularity allows changes to be made to code and data in a single self-contained DLL shared by several applications without the need to change the individual applications. In addition, modularity uses generic interfaces for plug-ins. A single interface allows both old and new modules to be integrated seamlessly at run-time into existing applications, without the need to modify the application itself.

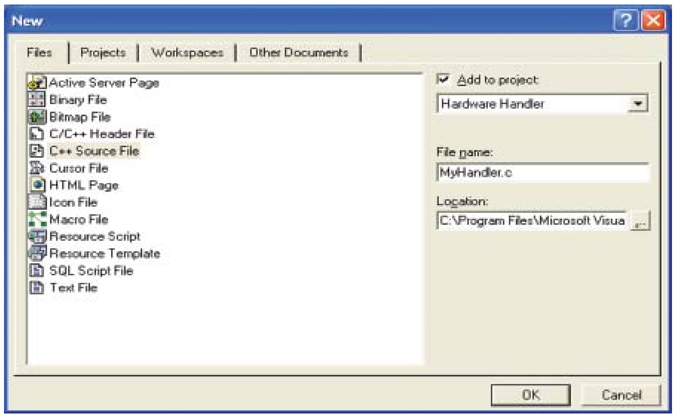
Selecting an empty DLL project will result in Win 32 creating a new skeleton project without any files added into the project.

Once an empty project is created, you need to specify the project settings. First, go to the 'General' tab and select 'Non Using MFC' since the Microsoft Foundation Classes (MFC) functionality is not used. Next, go to the 'Link' tab and specify 'utacore.lib' in the 'Options/library modules' column. The 'utacore.lib' must be linked to utilize the TestExec's functions. The next step is to create the implementation file as shown in Figure 4. Here, you have the option of selecting the file type.



Project name: MyHandler
Location: C:\Program Files\Microsoft Visua
[✓ Create new workspace]
Add to current workspace
Dependency of: Hardware Handler
Platforms: [✓ Win32]

Figure 3. Create a DLL project



[✓ Add to project:] Hardware Handler
File name: MyHandler.c
Location: C:\Program Files\Microsoft Visua

Figure 4. Create Implementation File

Step 3: Writing the Hardware Handler

This example outlines sample codes showing how an action written in C can communicate with the Agilent 34411 via the IVI-COM driver. Other TestExec SL's functionalities can also be added accordingly.

This example assumes that you have basic knowledge on TestExec SL's hardware API and functions. API functions or data types with 'UTA' in their names are based on Agilent's TestCore services which provide an open, standardized framework for creating or modifying test systems. Thus, 'UTA' or 'Uta' symbolizes the prefix that indicates a TestExec SL's function. You must also get familiarized with the TestExec SL's macros such as 'UTADLL' or 'UTAAPI'. These macros enhance code portability across operating platforms. 'UTAAPI' is used when declaring the function's prototype in a header file (.h) whereas 'UTADLL' is used when calling the function in an implementation file (.c or .cpp).

Initialization of Functions

Firstly, you should add the various header files for TestExec SL's interface declarations, the IVI-COM driver and various definitions used in writing the handler.

Here are examples of standard handler header files:

```
#include "stduta.h"
#include "common.h"
#include "HwhUtil.h"
#include "Dmm.h"
```

Here are examples of IVI-COM files:

```
#import "IviDriverTypeLib.dll"
#import "IviDmmTypeLib.dll"
#import "GlobMgr.dll"
#import "Ag34410.dll"
```

Writing Functions

The first section will cover the 4 mandatory functions which are Init (), Reset (), Declare Parms () and Reset (). These functions are required in every hardware handler. Subsequently, there will be two examples on controlling the DMM - measure DC voltage and configure the DMM.

A. Init ()

This function initializes a hardware module and is compulsory in all hardware handlers. When a testplan is executed, TestExec SL calls this function to initialize each instance of a hardware module that uses this hardware handler. An error handling is added to alert the user if the instrument fails to initialize as shown in the example here.

```
LPVOID UTADLL Init (HUTAHWMOD hModule, HUTAPB hParmBlock)
{
  IUtaString   szVISAaddr (hParmBlock, RESOURCE_STRING);

  INSTSTATE*  userHandle;
     userHandle = new(INSTSTATE);

  if (!userHandle){
     raise_exception (UTA_INST_ERROR_MESSAGE, "InstOpen", "Could
     not allocate memory for INSTSTATE", "");
     return NULL;
  }

  userHandle->vi            = (ViSession) 0;
  userHandle->ulHandlerID = NEWDMM_ID;
  userHandle->pInstState   = (C34411*)new(C34411);

  if (!(userHandle->pInstState)){
     raise_exception (UTA_INST_ERROR_MESSAGE, "InstOpen", "Could
     not allocate memory for new instance of AG34411A Dmm.", "");
     free(userHandle);
     return NULL;
  }

  if (!((C34411*)(userHandle->pInstState))-> Initialize
  (szVISAaddr,0,0,""))
  {
     raise_exception (UTA_INST_ERROR_MESSAGE, "InstOpen", "AG34411A
  Dmm.", "");
     delete (userHandle->pInstState);
     free(userHandle);
     return NULL;
  }

}
```

B. DeclareParms()

This function is used to declare parameters that TestExec passes to the DLL that contains the hardware handler. These parameters instruct the DLL on the exact hardware module to use. This is important in cases where the same DLL is used with more than one module of the same type. The variable 'DEVICEIDLABEL' is used to identify the device type.

```
  void UTADLL DeclareParms (HUTAHWMOD hModule, HUTAPBDEF hPBDef)
  {
  const char* const INT32TYPE = "CUtaInt32";
  HUTAINT32  hData = NULL;
  HUTASTRING hSData = NULL;

  hSData = (HUTASTRING)UtaHwModDeclareParm(hModule, hPBDef,
  DEVICEIDLABEL, "CUtaString", "E1411, HP34401A, AG34980A,
  AG34411A or L4411A only");
  UtaStringSetValue(hSData, VXI_DMM_1411);
  }
```

C. Close ()

This function closes a hardware module opened with the Init() function and is compulsory in all hardware handlers. Similar to the Init () function, TestExec SL calls this function for each instance of a hardware module that uses this hardware handler when a testplan is executed. This function is called when it is time to close the hardware module such as when TestExec SL exits or when system configuration changes. It is important to note that TestExec SL only calls this function if the hardware module was opened by a successful call to the Init () function, that is when no exceptions are raised. You have the option to implement this function in any way that is needed to close the hardware module such as freeing or deleting any memory associated with structures created with the Init () function.

```
void UTADLL Close (HUTAHWMOD hModule, HUTAPB hParmBlock,
LPVOID pInitData)
{
  HUTAINST      hInst = UtaHwModGetInst (hModule);
  INSTSTATE *   userHandle = (INSTSTATE *) pInitData;

  try
  {
     ((CDmm*)(userHandle->pInstState))->Close();
     delete (userHandle->pInstState);
     free (userHandle);
     return;
  }
  catch(...){
     raise_exception (UTA_INST_ERROR_MESSAGE, "Close", "Internal
     error in hwhdmm_base.dll.", "");
  }
}
```

D. Reset ()

This function resets a hardware module and returns the amount of time it takes to complete the reset process. This function is normally used when a new testplan is loaded or when recovery from an error is required. To prevent 'hot switching', TestExec resets the instruments first before resetting the hardware handler.

```
UTAUSECS UTADLL Reset (HUTAHWMOD hModule, HUTAPB hParmBlock,
LPVOID pInitData)
{

  IUtaInt32    nGlobalReset (hParmBlock, RSETLABEL);
  INSTSTATE *   userHandle = (INSTSTATE *) pInitData;

  if (nGlobalReset)
     ((CDmm*)(userHandle->pInstState))->Reset();

  return 0;
}
```

E. Voltage Measurement

This is an example on using Agilent 34411 to measure DC voltage via the action 'dmmMeasureDCV'.

```
void UTAAPI dmmMeasureDCV (HUTAINST dmm, SLONG meastype, double
expectedreading, long MeasurementMode, double *result)
{
  int nRtn;
  char* szInstErr;
  char szErr[5000];

  INSTSTATE *userHandle = NULL;
  userHandle = (INSTSTATE*) UtaInstGetUserHandle(dmm);
  CDmm* myDmm = (CDmm*)(userHandle->pInstState);

  if (myDmm == NULL){
     szInstErr = myDmm->GetError();
     sprintf(szErr, "Error in action dmmMeasureDCV.");
     UtaExcRaiseUserError(szErr, 9);
  }

  nRtn = myDmm->MeasDCVolts(meastype, expectedreading, result);
  if (!nRtn){
     szInstErr = myDmm->GetError();
     sprintf(szErr, "Error in action dmmMeasureDCV");
     UtaExcRaiseUserError(szErr, 9);
  }
}
```

If no error is detected, the function 'MeasDCVolts' in 'dmmMeasureDCV' will then invoke the function 'MeasDCVolts' in Class 34411 as shown in the example below. For the list of hardware functions that you can use to make reference calls, you can refer to the instrument's programming manual.

```
int C34411::MeasDCVolts(int resolution, double expected,
double *result)
{
  IAgilent34410DCVoltagePtr spDCV;
  try{
     spDCV = IAgilent34410Ptr(__uuidof(Agilent34410));
     *result = spDCV->Measure(range, resolution);
  }
  catch(...){
     sprintf (m_ErrStr, "Exception thrown in Agilent34411 IVI-COM
     driver IAgilent34411DCVoltagePtr interface.");
     return FN_FAIL;
  }
  return FN_SUCCESS;
}
```

F. DMM Configuration

This is an example on configuring the Agilent 34411 via the action 'dmmConfFunction'.

```
void UTADLL dmmConfFunction (HUTAINST dmm, SLONG func,
double range, double res)
{
  int nRtn;
  char* szInstErr;
  char szErr[5000];

  INSTSTATE *userHandle = NULL;
  userHandle = (INSTSTATE*) UtaInstGetUserHandle(dmm);
  CDmm* myDmm = (CDmm*)(userHandle->pInstState);

  if (myDmm == NULL){
     szInstErr = myDmm->GetError();
     sprintf(szErr, "Error in action dmmConfFunction.");
     UtaExcRaiseUserError(szErr, 9);
  }

  nRtn = myDmm->ConfigureFunction(func, range, res);
  if (!nRtn){
     szInstErr = myDmm->GetError();
     sprintf(szErr, "Error in action dmmConfFunction.");
     UtaExcRaiseUserError(szErr, 9);
  }
}
```

Step 4: Building the Project

After you have completed writing the hardware handler, it is good to verify the contents of your project to ensure that all the files are intact. Selecting the FileView pane helps you to check on your files easier. There are two configurations that you can choose to build your project – Release and Debug. The debug version contains additional codes that provide more details to facilitate your debug.

Conclusion

There are clear benefits of using DLLs. Firstly, DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily. In addition, DLLs help reduce memory overhead when several applications use the same functionality at the same time because although each application receives its own copy of the DLL data, the applications share the DLL code.

You can refer to the TestExec SL Online Help for the list of TestExec functions or if you have TS5400 library installed in your system, you can refer to the TS5400 Online Help for the list of hardware handler functions.

For more information on TestExec SL, please go to

www.agilent.com/find/testexec

www.agilent.com
www.agilent.com/find/testexec

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent Direct

www.agilent.com/find/agilentdirect
Quickly choose and use your test equipment solutions with confidence.

LXI

www.lxistandard.org
LXI is the LAN-based successor to GPIB, providing faster, more efficient connectivity. Agilent is a founding member of the LXI consortium.

Remove all doubt

Our repair and calibration services will get your equipment back to you, performing like new, when promised. You will get full value out of your Agilent equipment throughout its lifetime. Your equipment will be serviced by Agilent-trained technicians using the latest factory calibration procedures, automated repair diagnostics and genuine parts. You will always have the utmost confidence in your measurements. For information regarding self maintenance of this product, please contact your Agilent office.

Agilent offers a wide range of additional expert test and measurement services for your equipment, including initial start-up assistance, onsite education and training, as well as design, system integration, and project management.

For more information on repair and calibration services, go to:

www.agilent.com/find/removealldoubt

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas

Canada	(877) 894-4414
Latin America	305 269 7500
United States	(800) 829-4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe & Middle East

Austria	01 36027 71571
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	07031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland	0800 80 53 53
United Kingdom	44 (0) 118 9276201

Other European Countries:
www.agilent.com/find/contactus

Revised: July 2, 2009

Product specifications and descriptions in this document subject to change without notice.

Agilent Technologies